Preparing WL-LSMS for First Principles Thermodynamics Calculations on Accelerator and Multicore Architectures

OLCF
Oak Ridge Leadership Computing Facility

Markus Eisenbach

Oak Ridge National Laboratory

Motivation

- Density Functional Calculations have proven to be a useful tool to study the ground state of many materials.

- For finite temperatures the situation is less ideal an one is often forced to rely on model calculation with parameters either fitted to first principles calculations or experimental results.

- Fitting to models is especially unsatisfactory in inhomogeneous systems, nanoparticles or other systems where the model parameters could vary significantly from one site to another.

Solution:
Combine First Principles calculations with statistical mechanics methods

Thermodynamic Observables

- **Thermodynamic observables are related to the partition function Z and free energy F**

$$Z(\beta) = \sum_{\{\xi_i\}} e^{-\beta H(\{\xi_i\})}$$

$$F(T) = -k_B T \ln Z(1/k_B T)$$

- **If we can calculate Z(β) thermodynamic observables can be calculated as logarithmic derivatives.**

Wang-Landau Method

- Conventional Monte Carlo methods calculate expectation values by sampling with a weight given by the Bolzmann distribution
- In the Wang-Landau Method we rewrite the partition function in terms of the density of states which is calculated by this algorithm

$$Z(\beta) = \sum_{\{\xi_i\}} e^{-\beta H(\{\xi_i\})} = g_0 \int g(E) e^{-\beta E} dE$$

- To derive an algorithm to estimate g(E) we note that states are randomly generated with a probability proportional to 1/g(E) each energy interval is visited with the same frequency (flat histogram)

Metropolis Method

Metropolis et al, JCP **21**, 1087 (1953)

$$Z = \int e^{-E[\mathbf{x}]/k_{\mathrm{B}}T} d\mathbf{x}$$

Compute partition function and other averages with configurations that are weighted with a Boltzmann factor

Sample configuration where Boltzmann factor is large.

1. Select configuration

$$E_i = E[\mathbf{x}_i]$$

2. Modify configuration (move)

$$E_f = E[\mathbf{x}_f]$$

3. Accept move with probability

$$A_{i\to f} = \min\{1, e^{\beta(E_i - E_f)}\}$$

Wand-Landau Method

Wang and Landau, PRL **86**, 2050 (2001)

$$Z = \int W(E) e^{-E/k_{\mathrm{B}}T} dE$$

If configurations are accepted with probability $1/W$ all energies are visited equally (flat histogram)

1. Begin with prior estimate, eg $W'(E) = 1$
2. Propose move, accepted with probability

$$A_{i\to f} = \min\{1, W'(E_i)/W'(E_f)\}$$

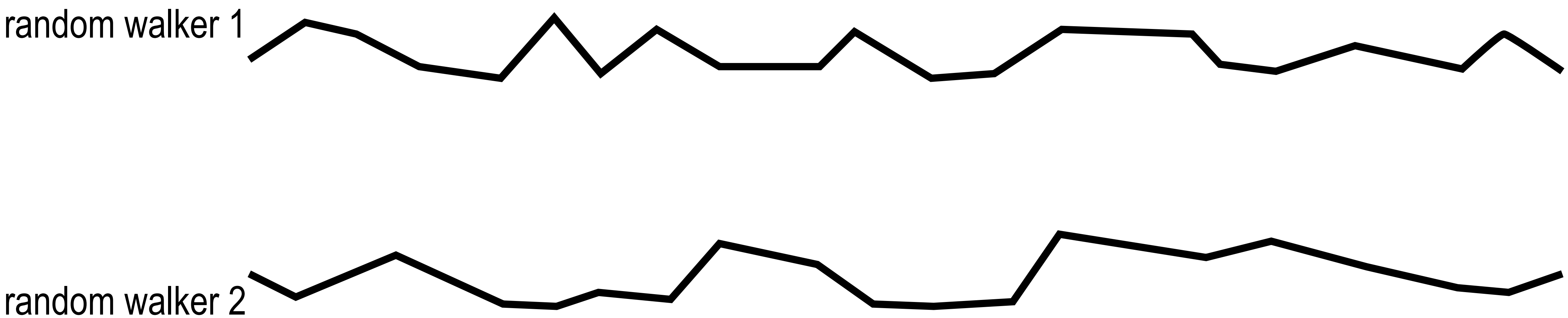
3. If move accepted increase DOS

$$W'(E_f) \to W'(E_f) \times f \qquad f > 1$$

4. Iterate 2 & 3 until histogram is flat
5. Reduce $f \to f = \sqrt{f}$ and go back to 1

Not quite embarrassingly parallel

Metropolis MC acceptance: $$A_{i \to f} = \min\{1, e^{\beta(E_i - E_f)}\}$$



Not quite embarrassingly parallel

Metropolis MC acceptance: $$A_{i\to f} = \min\{1, e^{\beta(E_i - E_f)}\}$$

Wang-Landau acceptance:

$$A_{i\to f} = \min\{1, e^{\alpha(w_\alpha(x_f) - w_\alpha(x_i))}\}$$

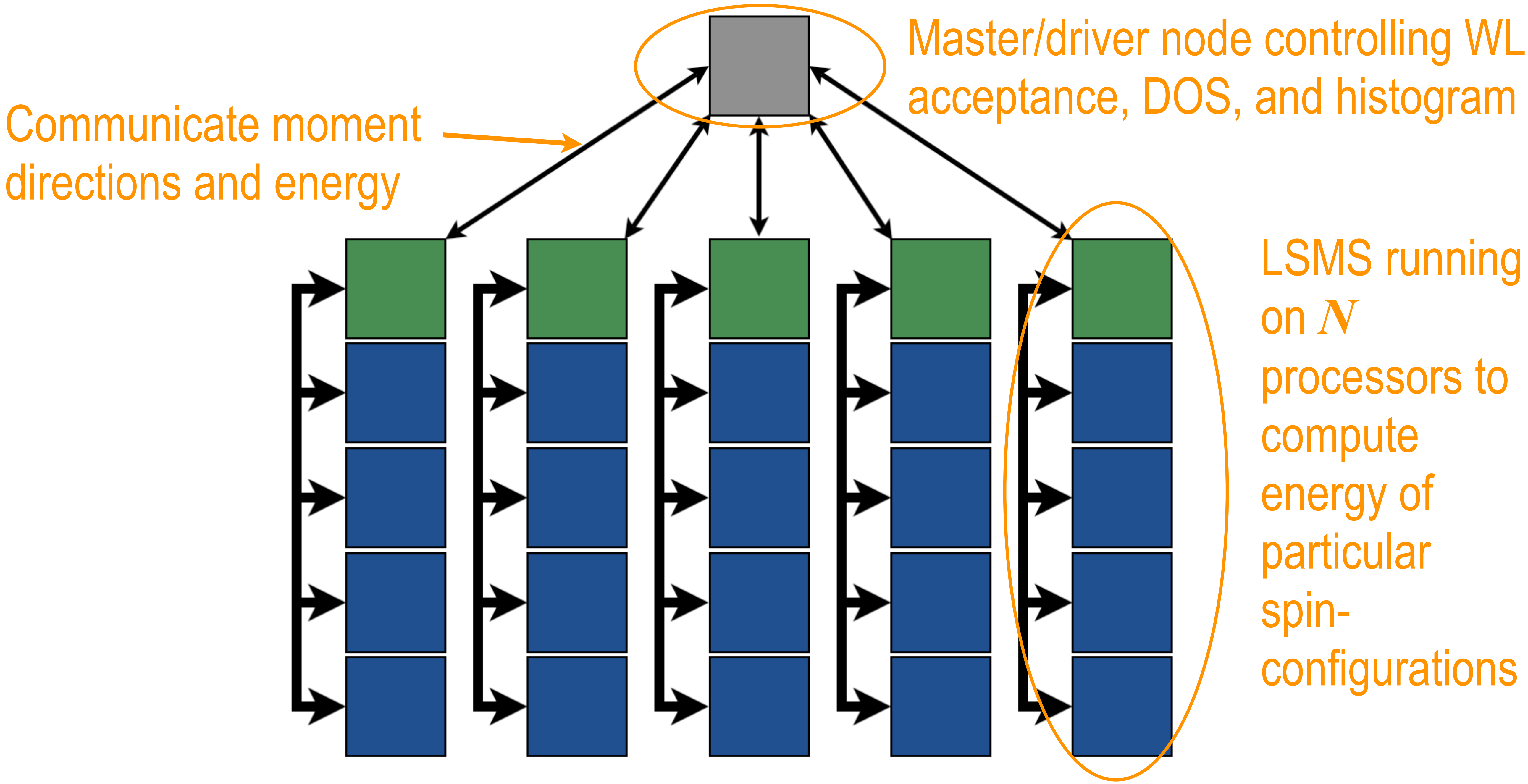
Not quite embarrassingly parallel

Metropolis MC acceptance: $$A_{i\to f} = \min\{1, e^{\beta(E_i - E_f)}\}$$

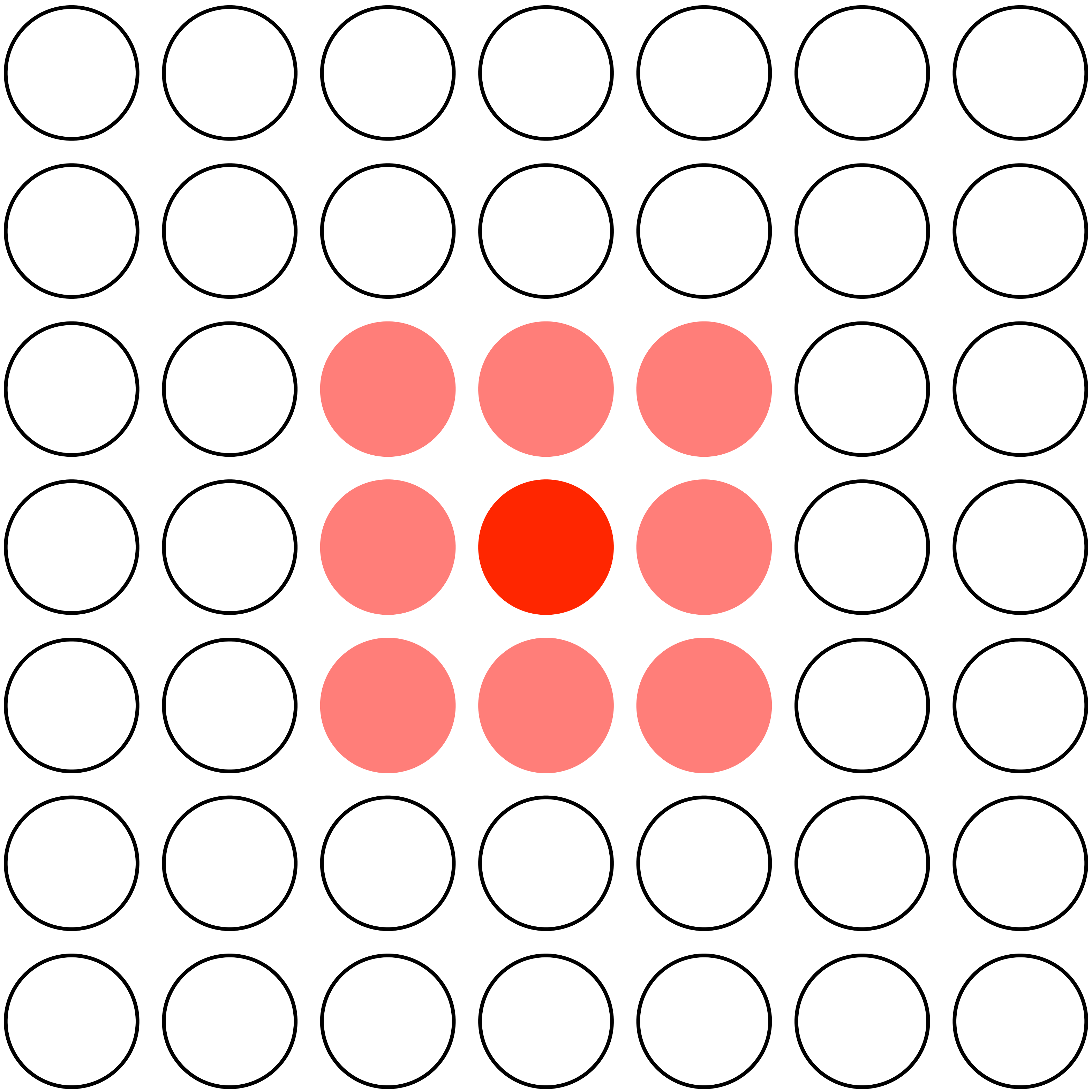
Wang-Landau acceptance:

$$A_{i\to f} = \min\{1, e^{\alpha(w_\alpha(x_f) - w_\alpha(x_i))}\}$$

Organization of the WL-LSMS code using a master-slave approach

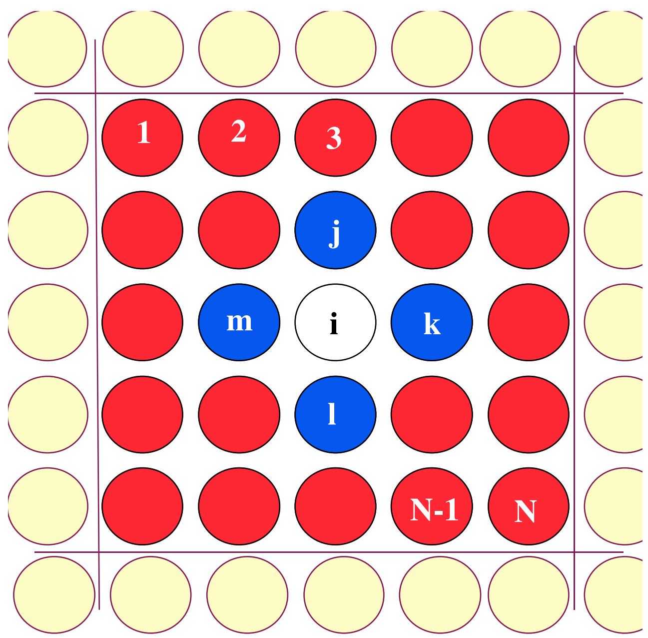


Nearsightedness and the locally self-consistent multiple scattering (LSMS) method

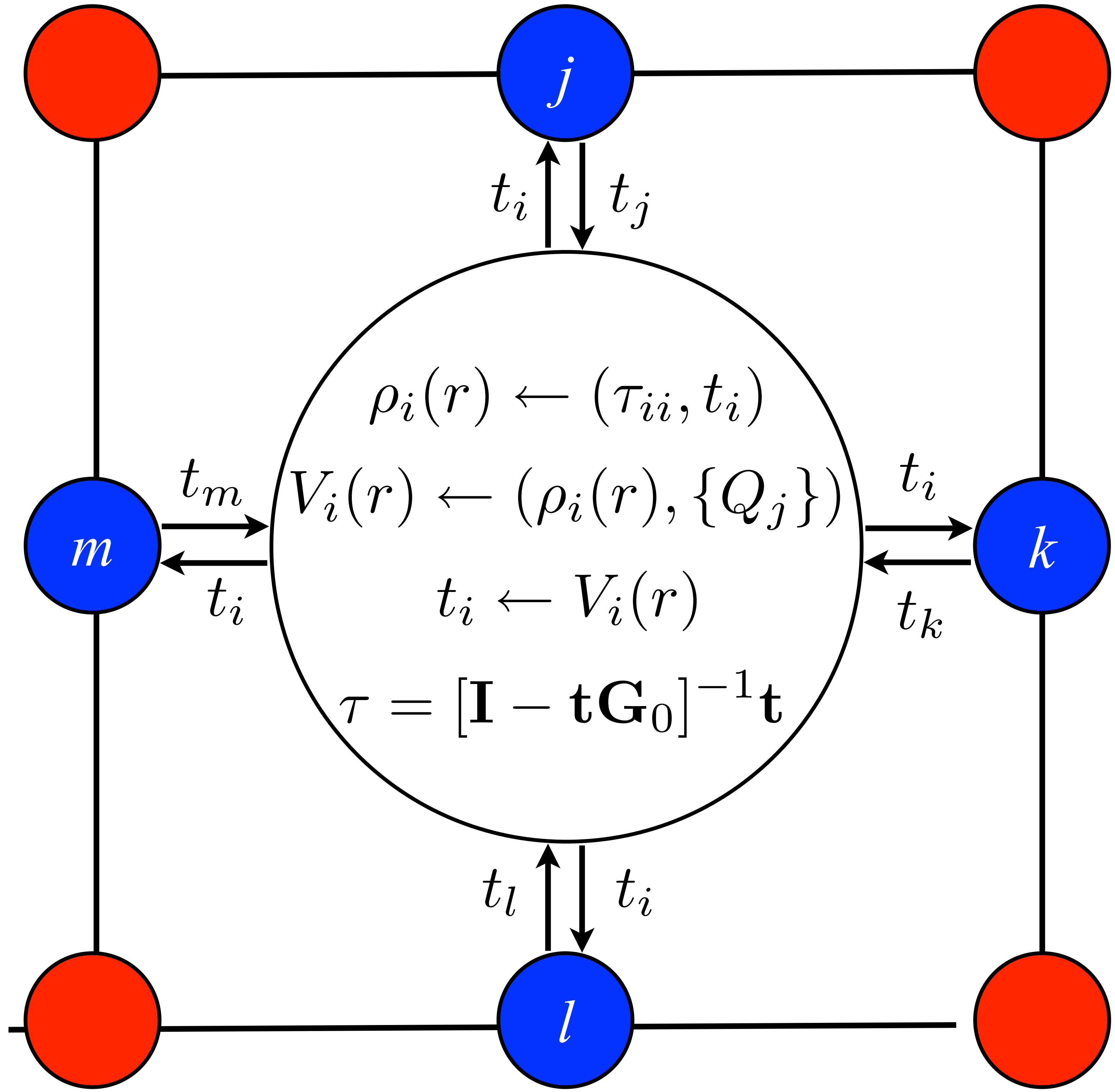


- Nearsightedness of electronic matter - Prodan & Kohn, PNAS **102**, 11635 (2005)
  - *Local electronic properties such as density depend on effective potential only at nearby points.*
- Locally self-consistent multiple scattering method - Wang et al., PRL **75**, 2867 (1995)
  - *Solve Kohn-Sham equation on a cluster of a few atomic shells around atom for which density is computed*
  - *Solve Poisson equation for entire system - long range of bare coulomb interaction*

A parallel implementation and scaling of the LSMS method: perfectly scalable at high performance



- Need only block i of τ
- $\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)^{-1} = \left(\begin{array}{c|c} (A - BD^{-1}C)^{-1} & * \\ \hline * & * \end{array}\right)$
- Calculation dominated by ZGEMM
- Sustained performance similar to Linpack



Refactoring LSMS_1 to LSMS_3

- LSMS_1 assumes one atom / MPI rank
  - But: This might not be ideal with current and future multicore CPU
  - Highly impractical for accelerators (CPUs)

- Increase flexibility of the code to adapt to new architectures and new physics

- Reduce the amount of code that needs to be rewritten
  - (This is essentially a one person effort)

- LSMS_1:

  Fortran (mainly 77) for LSMS

  C++ for Wang-Landau

LSMS_3

- Multiple atoms / MPI rank
  - possibility for multithreading (OpenMP) in LSMS
  - enable efficient use of accelerators

- New (less rigid) input file format

- Retain Wang-Landau part form LSMS_1

- LSMS_3:

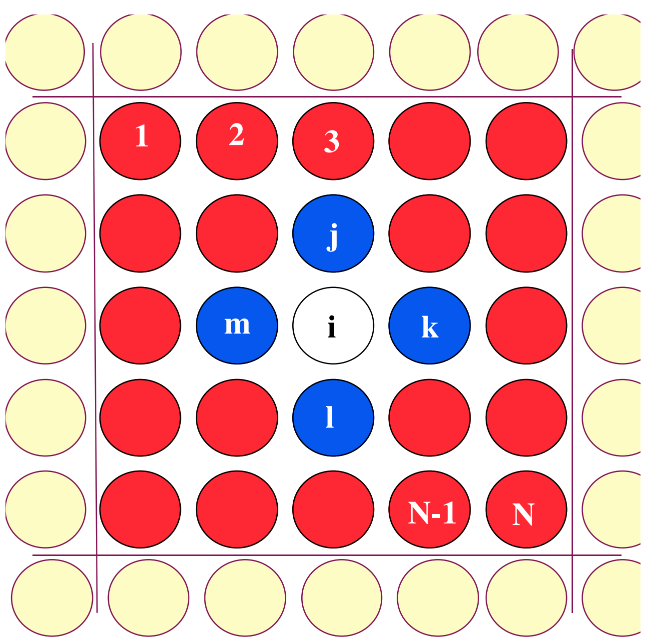
  Top level routines and data structures: C++
  New communication routines: C++
  Many compute routines from LSMS_1: Fortran

```
  LSMSSystemParameters lsms;
  LSMSCommunication comm;
  CrystalParameters crystal;
  LocalTypeInfo local;

// Initialize communication and accelerator

// Read the input file

  communicateParameters(comm,lsms,crystal);

  local.setNumLocal(distributeTypes(crystal,comm));
  local.setGlobalId(comm.rank,crystal);

  buildLIZandCommLists(comm,lsms,crystal,local);

  loadPotentials(comm,lsms,crystal,local);

  setupVorpol(lsms,crystal,local,sphericalHarmonicsCoeficients);

  calculateCoreStates(comm,lsms,local);

  energyContourIntegration(comm,lsms,local);
  calculateChemPot(comm,lsms,local,eband);
```

Multiple Atoms / MPI rank

An important step to enable efficient use of multicore and accelerator architectures: Allow for **more** work / MPI rank!

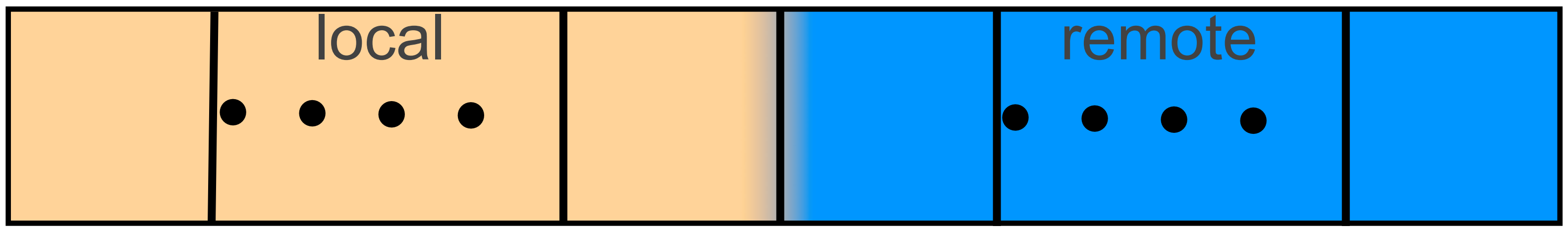
In LSMS: multiple atoms / MPI rank
necessitates new communication pattern



for all atoms i in the crystal **do**
  build the local interaction zone $\mathrm{LIZ}_i = \{j | dist(\mathbf{x}_i, \mathbf{x}_j) < r_{\mathrm{LIZ}}\}$ of atom i
  for all atoms j in LIZ_i **do**
    add atom j to the list R_i of data to receive for atom i (`tmatFrom`)
    add atom i to the list S_j of data to send from atom j (`tmatTo`)
  end for
end for
remove duplicate entries from S_j and R_i

Multiple Atoms / MPI rank

```
Matrix<Complex> tmatStore;
```



t matrices needed for building the local tau matrices

Building the tau matrices:

(1) Prepost receives for remote t matrices
(2) Loop over all local atom (OpenMP)
    calculate local t matrices
(3) Send local t matrices
(4) wait for completion of communication

```
expectTmatCommunication(comm,local);

for(int i=0; i<local.atom.size(); i++)
  calculateSingleScattererSolution(lsms,local.atom[i],vr[i],energy,prel,pnrel, solution[i]);

sendTmats(comm,local);
finalizeTmatCommunication(comm);
```

Calculating the tau matrix $\tau = \left[I - tG_0\right]^{-1} t$

```
for(int i=0; i<local.num_local; i++)
    calculateTauMatrix(lsms,local,local.atom[i],
                       energy,prel,&tau00_l(0,i));
```

(1) For all local atoms (possibility for multithreading)
- (a) build m matrix (m=I-tG) (multithreading or accelerator)
- (b) invert m matrix (multithreading or accelerator)
- (c)

$$\tau = \left[I - tG_0\right]^{-1} t$$

m has rank k * #LIZ and can be broken in k * k blocks m_{ij}

$$m_{ij} = I\delta_{ij} - t_i G_0^{ij}$$

in most cases only the diagonal block for the local site is needed

$$\tau_{00} = (m^{-1})_{00} t_0$$

Block Inverse

The LSMS method requires only the first diagonal block of the inverse matrix

Recursively apply

$$\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)^{-1} = \left(\begin{array}{c|c} (A - BD^{-1}C)^{-1} & * \\ \hline * & * \end{array}\right)$$

The block size is a performance tuning parameter:

- Smaller block size: less work
- Larger block size: higher performance of matrix-matrix multiply

Performance of LSMS dominated by double complex matrix matrix multiplication

`ZGEMM`

Main zblock_lu loop
BLAS: CPU, LAPACK: CPU

```
        n=blk_sz(nblk)
        joff=na-n
        do iblk=nblk,2,-1
        m=n
        ioff=joff
        n=blk_sz(iblk-1)
        joff=joff-n
  c invert the diagonal blk_sz(iblk) x blk_sz(iblk) block
        call zgetrf(m,m,a(ioff+1,ioff+1),lda,ipvt,info)
  c calculate the inverse of above multiplying the row block
  c blk_sz(iblk) x ioff
        call zgetrs('n',m,ioff,a(ioff+1,ioff+1),lda,ipvt,
       &     a(ioff+1,1),lda,info)
        if(iblk.gt.2) then
        call zgemm('n','n',n,ioff-k+1,na-ioff,cmone,a(joff+1,ioff+1),lda,
       &     a(ioff+1,k),lda,cone,a(joff+1,k),lda)
        call zgemm('n','n',joff,n,na-ioff,cmone,a(1,ioff+1),lda,
       &     a(ioff+1,joff+1),lda,cone,a(1,joff+1),lda)
        endif
        enddo
        call zgemm('n','n',blk_sz(1),blk_sz(1)-k+1,na-blk_sz(1),cmone,
       &     a(1,blk_sz(1)+1),lda,a(blk_sz(1)+1,k),lda,cone,a,lda)
```

Main zblock_lu loop
BLAS: CPU
LAPACK: CPU

```
do iblk=nblk,2,-1
...

call zgetrf(…)
call zgetrs(…)

call zgemm(…)
call zgemm(…)

enddo

call zgemm(…)
```

Main zblock_lu loop – GGD
BLAS: GPU (CUDA)
LAPACK: GPU (CULA device API)

```
call cublas_set_matrix(…)

do iblk=nblk,2,-1
...

call cula_device_zgetrf(…)
call cula_device_zgetrs(…)

call cublas_zgemm(…)
call cublas_zgemm(…)

enddo

call cublas_zgemm(…)

call cublas_get_matrix(…)
```

WL-LSMS3

- **First Principles Statistical Mechanics of Magnetic Materials**
- **identified kernel for initial GPU work**
  - zblock_lu (95% of wall time on CPU)
  - kernel performance: determined by BLAS and LAPACK: ZGEMM, ZGETRS, ZGETRF
- **preliminary performance of zblock_lu for 12 atoms/node of Jaguarpf or 12 atoms/GPU**
  - For Fermi C2050, times include host-GPU PCIe transfers
  - Currently GPU node does *not* utilize AMD Magny Cours host for compute

	Jaguarpf node (12 cores AMD Istanbul)	Fermi C2050 using CUBLAS	Fermi C2050 using Cray Libsci
Time (sec)	13.5	11.6	6.4

Slide provided by Cray